##

Naslov: Osnove C prorgramiranja
Autor: sp1r1t
Mail: resistant89@hotmail.com
Maj 2009.

SADRZAJ:

|-- O C-u
|-- Osnove
|-- Kompajliranje
|-- Tipovi Funkcija
|-- Komentari
|-- Pocetak
|-- Tipovi podataka
|-- Operatori u C-u
|-- Definisanje konstanti
|-- Funkcija printf()
|-- Start(krecemo sa pisanjem programa)
|-- Makro zamena
|-- Funkcija unosa scanf()
|-- IF naredba grananja
|-- Logicki operatori
|-- Operatori ++ i --
|-- Operatori dodele
|-- SWITCH - Operator visestrukog izbora
|-- WHILE i DO WHILE ciklusi
|-- Operator ciklusa FOR
|-- Naredba BREAK
|-- Funkcije
|-- Outro

Programski jezik C, napravio neki lik po imenu Dennis Ritchie, 1972. godine
za koriscenje na unix sistemima. Iako je napravljen za sistemsko programiranje
takodje se koristi i za programiranje aplikacija.

Prvi programski jezik koji sam naucio, C, nie tezak za ucenje, jednostavna
sintaksa sve u svemu kul jezik za pocinjanje sa programiranjem.
U danasnje vreme, uglavnom se koristi za sistemsko programiranje, mada se moze
naci i raznim granama.

OSNOVE:

Pre bilo kog programskog jezika, treba nauciti osnove programiranja, sta je to
racunar uopste, kako sve to funkcionise itd. Kada ovo naucimo, bice nam lakse
da ustanovimo sta u stvari mozemo, a sta ne mozemo programiranjem da uradimo.

Svaki C program MORA da sadrzi 2 stvari:

1) biblioteke
2) main() funkciju

Biblioteke su fajlovi u kojima se nalaze neke napisane funkcije koje programer
moze da koristi kako ne bi morao da sam pise te funkcije (npr: stampanje teksta
na standardni izlaz (monitor), funkcija printf).

Kod vecine kompajlera ili IDE-a (integrated development environment), u koliko se u source ne upisu biblioteke, on automatki dodaje osnovne a to su uglavnom stdio.h i stdlib.h .

main() funkcija je funkcija koju program prvu izvrsava, tacnije samo nju i zna da izvrsi, tako da, u koliko nemamo main() funkciju, program nece znati sta da pokrene.

KOMPAJLIRANJE:

Dok programiramo, mi ne pravimo izvrsnu verziju programa, vec samo source fajl, koji je u C-u fajl.c .Da bi od ovog source fajla napravili izvrsni fajl (exe) potreban nam je kompajler. I za Windows i za Linux platforme postoji dosta programa za kompaliranje. Ja cu navesti samo neke koje sam koristio.

Windows:
Turbo C - fin program, za editovaje i kompaliranje, DOS okruzejne, sto ce biti malo ruzno pocetnicima, ali znajte da je C veoma star jezik i da kada je on napravljen, nije postojo GUI, tako da, imajte postovanja.
http://edn.embarcadero.com/article/20841

Code::Blocks - program koji ja koristim, freeware, lepo izgleda, jednostavan za koriscenje, koristi GCC kompajler, takodje, koristi se za C i C++. Preporucio bih ga onima koji planiraju da se bave C programiranjem.
http://www.codeblocks.org/

Linux:
GCC - Nisam neki vrstan korisnik Linuxa ali znam tu i tamo ponesto, tako da standardni kompajler, bez koga linux nije linux, takozvani GCC.
http://gcc.gnu.org/

Code::Blocks - da, code blocks postoji i za linux, vec sam ispricao o njemu, tako nema potrebe da ponavljam. Evo jos jednom link.
http://www.codeblocks.org/
FUNKCIJE:

Da krenemo od funkcija, funkcija se definise imenom, zatim se u zagrade ubacuju parametri funkcije i telo funkcije zapocinje znakom { a zavrsava sa } .

Primer funkcije (prazne)

```
--------------------------------------------------------------------------------
main() {
}
--------------------------------------------------------------------------------
```

TIPOVI FUNKCIJA:

Tipovi funkcije int i void: dakle, u zavisnosti od rezultata koji funkcija vraca delimo funkcije na int i void i to na sledeci nacin: u koliko funkcija vraca ceo broj, funkcija je int, u koliko funkcija izvrsava neku operaciju i nije nam bitno sta vraca, ili funkcija ne vraca nista, onda je funkcija void. Funkciju main() ne moramo definisati jer ona ne vraca nikakvu vrednost, mada se to moze raditi, bice prikazano i naglaseno u toku teksta.
NOTE: ako nam je funkcija int, ona mora vracati neki ceo broj!

KOMENTARI:

Komentari se u C-u obelezavaju na sledeci nacin:
komentar pocinje simbolom /* a zavrsava se obrnutim, dakle */ ,evo primera komentara

```
/* ovo je moj komentar
koji moze da ide u vise redova
i trajace sve dok ne stavim tag za zatvaranje komentara
*/
```

Jedna od napomena, svaku liniju unutar funkcija, svaku komandu, moramo zavrsiti sa znakom ; ovaj znak oznacava kraj komande i program moze krenuti sa izvrsavanjem sledece, u koliko nema ; program ce spajati komande i prijavljivati sintaksne greske. Komande koje ne treba zavrsavati sa znakom ; jesu petlje: for, while i if.

Programski jezik C je case sensitive sto znaci da je 'string' razlicito od 'String' ili 'STRING'.

U C-u postoje rezervisane reci koje se ne mogu koristiti kao imena promenljivih ili nesto slicno, a to su npr if, for, while, int, void...

POCETAK:

Krecemo sa pisanjem prvog programa:

```
#include <stdio.h>
main() {
printf("Hello World!");
}
```

Kao sto sam spomenuo kod tipova funkcija, ovaj isti kod mozemo napisati i ovako:

```
#include <stdio.h>
int main() {
printf("Hello World!");
return 0;
}
```

Objasnjenje:

```
#include <stdio.h>     <- ukljucivanje ili inkludovanje biblioteke stdio.h u nas program
main() {     <- pocetak funkcije main()
printf("Hello World!");     <- pozivanje funkcije printf() sa parametrima pod navodnicima unutar zagrada, u nasem primeru "Hello World!"
}     <- kraj main() funkcije
```

Da ne bi objasnjavao iste redove, objasnicu samo redove koji se razlikuju od prethodnog primera.

```
int main() {     <- pocetak funkcije main() i definisanje da vraca ceo broj
return 0;     <- vracamo 0 sistemu, sto u main() funkciji oznacava da je sve izvrseno kako treba i da nema nikakvih problema
```

Takodje da dodam, kao dodatni parametri u funkciji printf() se koriste i takozvane komande. Primer komande je recimo za novi red a to je \n ili za TAB razmak \t

Primer:

```
--------------------------------------------------------------------------------
main() {
printf("dobar dan\n");
printf("\n");
printf("ovo je tekst o osnovama C-a\t");
printf("vreme je 6:00");
}
--------------------------------------------------------------------------------
```

Ova funkcija ce nam ekran izbaciti sledeci rezultat:

```
--------------------------------------------------------------------------------
dobar dan

ovo je tekst o osnovama C-a           vreme je 6:00
--------------------------------------------------------------------------------
```

TIPOVI PODATAKA U C-u:

celobrojni tip:
int
short
long
long long
unsigned
....

realni tip:
float
double

znakovni tip:
char

primeri definisanja varijabli(podataka) i dodeljivanja vrednosti

```
int a;
int broj=5;
float PI=3.14;
char znak;
char karakter = 'k';
```

OPERATORI U C-u:

Aritmeticki operatori:
+ - sabiranje
- - oduzimanje
* - mnozenje
/ - deljenje
% - ostatak pri deljenju

Operatori poredjenja:

```
>  - vece
>= - vece ili jednako
<  - manje
<= - manje ili jednako
== - jednako
!= - razlicito

Logicki operatori:
!  - negacija
&& - konjugacija (takodje poznato kao 'AND' ili 'i' operator)
|| - disjunkcija (poznato kao 'OR' ili 'ili' operator)

Operatori uvecanja ili umanjenja:
++  - uvecanje (postfiksno i prefiksno)
--  - umanjenje (postfiksno i prefiksno)

Operatori dodele:
=
op= (+=, -=, *= ...)

DEFINISANJE KONSTANTI:

Konstante definisemo na sledeci nacin:

#define <ime konstante> <njena vrednost>

Primer:

#define PI 3.14

FUNKCIJA PRINTF():

--------------------------------------------------------------------------------
#include <stdio.h>
#define PI 3.14
main () {
printf("Broj PI ima vrednost %f\n", PI);
}
--------------------------------------------------------------------------------

Specifikacije konverzije:

%c - char
%d - int
%u - ceo dekadni broj bez znaka
%o - oktalni broj bez znaka
%x - heksadekadni broj bez znaka
%s - string
%f - float, double

Dakle, sada kada smo objasnili specifikacje, da objasnimo prethodni primer. Uzecu
samo bitne delove koda.

#define PI 3.14
printf("Broj PI ima vrednost %f\n", PI);

Sa %f smo oznacili da tu ide neka vrednost koja je double ili float (realan broj),
```

a nakon zareza ',' smo oznacili koji broj se smesta na to mesto.

START(krecemo sa pisanjem programa):

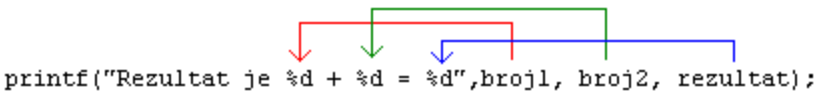
Ovo su bile neke osnove, i kada smo njih naucili, mozemo krenuti sa pisanjem prvih programa. Za pocetak cemo napisati program koji racuna zbir dva broja.

```
#include <stdio.h>
main() {
int broj1, broj2, rezultat;
broj1 = 10;
broj2 = 20;
rezultat = broj1 + broj2;
printf("Rezultat je %d + %d = %d",broj1, broj2, rezultat);
}
```

Kada pokrenemo program, dobicemo sledeci ispis na ekranu:

```
Rezultat je 10 + 20 = 30
```

U ovom primeru imamo deklarisanje varijabli (int broj1, broj2..), dodeljivanje vrednosti varijabli (broj1 = 10;), operacije dodeljivanja i aritmeticku operaciju sabiranja (rezultat = broj1 + broj2;) i odlican primer koriscenja printf() funkcije. U ovom primeru u printf() funkciji imamo 3 puta %d i nakon ',' 3 varijable kojima redom dodeljujemo mesta. Evo slike kako bi lakse shvatili:



Ovaj isti primer smo mogli da napisemo i krace, ali sam ga prvo napisao u prosirenom obliku kako bi lakse shvatili o cemu se radi, evo i u kracem obliku:

```
#include <stdio.h>
main() {
int broj1=10, broj2=20;
printf("Rezultat je %d + %d = %d",broj1, broj2, broj1+broj2);
}
```

Krecemo sa odnosom tipova. Evo primera (char i int):

```
#include <stdio.h>
main() {
int vrednost;
vrednost = 'A';
printf("%s\nkarakter = %c\nvrednost = %d\n", "veliko slovo", vrednost, vrednost);
vrednost = 'a';
printf("%s\nkarakter = %c\nvrednost = %d\n", "malo slovo", vrednost, vrednost);
}
```

```
--------------------------------------------------------------------------------
```

Ovaj program ce nam na ekran ispisati sledeci rezultat:

```
--------------------------------------------------------------------------------
veliko slovo
karakter = A
vrednost = 65
malo slovo
karakter = a
vrednost = 97
--------------------------------------------------------------------------------
```

U prvoj printf() funkciji, promenljiva ima vrednost 'A' i kada ispisujemo char tip promenljive, ipisuje se 'A', ali kada ispisujemo int vrednost (%d), ispisuje se njena ASCII vrednost sto je 65. Isto vazi i za drugu printf() funkciju. Char 'a', int 97.

MAKRO ZAMENA:

Pre smo u tekstu imali jedan primer makro zamene, ali nisam hteo da ga naglasavam sve do sada kada cu objasniti sta je to ustvari. Primer o kome sam govorio je ovaj

```
printf("Rezultat je %d + %d = %d",broj1, broj2, broj1+broj2);
```

Makro zamena je sledeci kod "broj1+broj2", evo jos jednog primera. Kilogram gumenih bombona je 15 evra, koliko iznosti 5 kilograma gumenih bombona

```
--------------------------------------------------------------------------------
#include <stdio.h>
#define GUMENE_BOMBONE 15
main() {
printf("5 kilograma gumenih bombona iznosi %d", 5*GUMENE_BOMBONE);
/* printf("1 kilogram gumenih bombona iznosi %d", GUMENE_BOMBONE); */
}
--------------------------------------------------------------------------------
```

FUNKCIJA UNOSA SCANF():

Funkcija scanf() nam sluzi za unos podataka sa tastature. Evo jednog primera kako bi lakse shvatili:

```
--------------------------------------------------------------------------------
#include <stdio.h>
main () {
int a,b;
printf("unesi dva boja za mnozenje\n");
scanf("%d %d", &a, &b);
printf("proizvod brojeva %d i %d je %d", a, b, a*b);
}
--------------------------------------------------------------------------------
```

Kada pokrenemo program, dobicemo sledecu poruku na ekranu:

```
unesi dva boja za mnozenje
```

nakon toga, program ce cekati da se unesu dva broja, unosimo u sledecem formatu

5 2

nakon sto upisemo ta 2 broja i pritisnemo enter, na ekranu ce se ispisati sledece:

proizvod brojeva 5 i 2 je 10

to sve izgleda ovako:

```
--------------------------------------------------------------------------------
unesi dva boja za mnozenje
5 2
proizvod brojeva 5 i 2 je 10
--------------------------------------------------------------------------------
```

IF NAREDBA GRANANJA:

Sintaksa:
```
if(uslov) naredba1; else naredba2;
```

U koliko imamo vise naredbi, potrebno ih je staviti unutar {} zagrada.

```
--------------------------------------------------------------------------------
if(uslov) {
naredba1;
naredba2;
}
else {
naredba3;
naredba4;
}
--------------------------------------------------------------------------------
```

Evo jednog malog i kratkog primera, kako bi lakse shvatili.

```
--------------------------------------------------------------------------------
#include <stdio.h>
main() {
int a,b;
printf("unesi dva broja\n");
scanf("%d %d", &a, &b);
if(a>b) printf("Broj %d je veci od broja %d", a, b);
else printf("Broj %d je veci od broja %d", b, a);
}
--------------------------------------------------------------------------------
```

Kod ovog primera postavljamo pitanje, sta ako unesemo dva ista broja. npr 5 i 5 program ce ispisati sledece "Broj 5 je veci od broja 5" sto nije tacno. Evo programa koji resava to, a i ujedno je primer if unutar if grananja.

```
--------------------------------------------------------------------------------
#include <stdio.h>
main() {
int a,b;
printf("unesi dva broja\n");
scanf("%d %d", &a, &b);
if(a==b) printf("Brojevi su jednaki");
```

```
else {
    if(a>b) printf("Broj %d je veci od broja %d", a, b);
    else printf("Broj %d je veci od broja %d", b, a);
}
}
```

Program prvo ispituje da li su dva broja jednaka, u koliko jesu, ispisuje da su jednaki, u koliko nisu, nastavlja proveru koji je veci.

Uslovni operator ?:

Ovo je u stvari skraceni oblik pisanja if-a koji se retko koristi, ali cu ga spomenuti za slucaj da ga sretnemo negde, pa da znamo o cemu se radi.

Sintaksa:
(uslov) naredba1 : naredba2;

Primer:

```
#include <stdio.h>
main() {
int a, b;
printf("unesi dva broja\n");
scanf("%d %d, &a, &b);
printf("veci broj je %d", (a>b) ? a : b);
}
```

U koliko u program unesemo brojeve 2 i 3, program ce izbaciti rezultat:

```
veci broj je 3
```

IF, ELSE IF, ELSE

Ako imamo vise uslova, pored if i else, koristimo i else if

```
if(uslov) naredba1;
else if(uslov2) naredba2;
else if(uslov3) naredba3;
else naredba4;
```

Evo primera. Program od malo pre koji racuna koji je od dva broja veci ukljucujuci ispitivanje da li su brojevi isti

---

```
#include <stdio.h>
main() {
int a,b;
printf("unesi dva broja\n");
scanf("%d %d", &a, &b);
if(a==b) printf("brojevi su jednaki");
```

```
else if(a>b) printf("%d je veci od %d", a, b);
else printf("%d je veci od %d", b, a);
}
```

LOGICKI OPERATORI(! && ||):

Prethodno smo ih u tekstu napomenuli, ali sada cemo ih objasniti i objasniti njihovo koriscenje.
Logicki operator za negaciju ili ! . Primer:

```
#include <stdio.h>
main() {
int broj=5;
if(broj!=5) printf("broj je razlicit od 5");
else printf("broj je jednak broju 5");
}
```

Ovde imamo jedno IF grananje u kome pise sledece, ako je broj razlicit od 5 (broj!=5), ispisi poruku da je razlicit, u suprotnom, ispisi poruku da je broj jednak broju 5.

Logicki operator za konjugaciju, i, AND, &&
Primer:

```
#include <stdio.h>
main() {
int n,m;
printf("unesi dva broja\n");
scanf("%d %d", &n, &m);
if(n==3 && m==3) printf("oba uneta broja su jednaka broju 3");
else printf("jedan ili ni jedan od unetih brojeva nije jednak broju 3");
}
```

Evo 2 primera izlaza (u zavisnosti od unosa) programa:

```
unesi dva broja
3 3
oba uneta broja su jednaka broju 3
```

---

i drugi:

```
unesi dva broja
2 3
jedan ili ni jedan od unetih brojeva nije jednak broju 3
```

Kao sto vidimo, moraju biti zadovoljena oba uslova da bi se ispisala poruka da su oba broja jednaka broju 3. A sada cemo objasniti kako napraviti program kome ce biti dovoljan samo jedan uslov od navedena dva.

Logicki operator za disjunkciju, ili, OR, ||
Primer:

```
#include <stdio.h>
main() {
int n,m;
printf("unesi dva broja\n");
scanf("%d %d", &n, &m);
if(n==3 || m==3) printf("jedan ili oba od unetih brojeva su jednaki broju 3");
else printf("ni jedan od brojeva koji je unet nije jednak broju 3");
}
```

Evo 2 primera izlaza programa, u zavisnosti od unetih brojeva:

```
unesi dva broja
3 8
jedan ili oba od unetih brojeva su jednaki broju 3
```

i drugi:

```
unesi dva broja
2 4
ni jedan od brojeva koji je unet nije jednak broju 3
```

OPERATORI ++ i --:

Operator ++ je operator uvecanja i oznacava isto sto i +1. primer:
x++; je isto sto i x = x + 1; . Ovo se isto odnosi i na operator umanjenja -- .

Postfiksni oblik:

Postfiksni oblik operatora npr. uvecanja je sledeci: x++; . Evo jednog primera kako bi lakse shvatili:

```
#include <stdio.h>
main() {
int x=5,y;
y = x++;
printf("x = %d", x);
printf("y = %d", y);
}
```

Program ce nam izbaciti sledece:

```
x = 6
y = 5
```

y = x++; <- u ovoj komandi, vrednost x se dodeljuje vrednosti y, a zatim se x uvecava za 1.

Prefiksni oblik:

Prefiksni oblik operatora npr. uvecanja je sledeci ++x; . Evo i primera:

```
---------------------------------------------------------------------------
#include <stdio.h>
main() {
int x=5,y;
y = ++x;
printf("x = %d", x);
printf("y = %d", y);
}
---------------------------------------------------------------------------
```

Program izbacuje sledeci rezultat:

```
---------------------------------------------------------------------------
x = 6
y = 6
---------------------------------------------------------------------------
```

Kao sto vidimo rezultat je drugaciji od prethodnog primera (postiksnog operatora) jer se u ovom primeru, vrednost x PRVO uveca za 1, a zatim se vrednost x dodeljuje vrednosti y. U tome je razlika postfiksnog i prefiksnog operatora.

OPERATORI DODELE:

Operatori dodele nam sluze za dodeljivanje odredjene vrednosti nekoj promenljivoj. Najosnovniji operator dodele je = (znak jednako). Evo primera kako bi lakse shvatili:
broj1 = broj2;
Ovom naredbom, vrednost iz promenljive broj2 smo smestili ili kopirali u promenljivu broj1. Operatori dodele su takodje aritmeticki operatori uz operator =, npr. +=, -=..
Evo primera:
zbir+=broj;
Ovo je ista naredba kao kad bi smo napisali ovo:
zbir=zbir+broj;
Da objasnim, sabira staru vrednost promenljive zbir sa vrednoscu promenljive broj i smesta je kao novu vrednost u promenljivu zbir. Npr ako nam je stara vrednost promenljive zbir 5, a vrednost promenljive broj 1, to ce izgledati ovako:

```
---------------------------------------------------------------------------
zbir=5
broj=1
zbir=5+1
zbir=6
---------------------------------------------------------------------------
```

Ovakav vid zapisa ce biti objasnjen kasnije u toku ovog teksta.
Operatori dodele +=, -=, *= i /= su u stvari samo skraceni oblici odredjenih naredbi i u koliko vas bune, mozete koristiti obican zapis, evo male tablice:

```
---------------------------------------------------------------------------
suma=suma+broj                  suma+=broj
```

```
suma=suma-broj                  suma-=broj
suma=suma*broj                  suma*=broj
suma=suma/broj                  suma/=broj
```

SWITCH - OPERATOR VISESTRUKOG IZBORA

Ovo je odlican operator kojim omogucujemo visestruko grananjei izvrsavanje razlicitih komandi. Switch se moze zameniti i sa vise if naredbi.

Sintaksa:

```
switch(izraz) {
             case konstanta1: naredba1; break;
             case konstanta2: naredba2; break;
             case konstanta3: naredba3; break;
default: naredba4;
```

Izraz koji stavljamo unutar switch(izraz) mora da bude ceo broj (int) ili char koji se interno konvertuje u int. Svaki case (slucaj) moramo zavrsiti komandom break; jer ce u suprotnom program izvrsavati redom ostale slucajeve, a mi to ne zelimo. Slucaj (case) default, oznacava komandu ili niz komandi koji ce se izvrsiti ako izraz ne ispunjava ni jedan od navedenih slucajeva, i posle njega ne moramo staviti break; . Slucaj default se moze izostaviti. Inace komanda break; se koristi za nasilno izlazenje iz komande ili operatora. Evo odma i primera switch operatora. Napisacemo program koji za uneti redni broj dana u nedelji ispisuje ime dana u nedelji, npr. 1 = ponedeljak, 2 = utorak...

```
#include <stdio.h>
main() {
int dan;
printf("Unesi redni broj dana u nedelji\n");
scanf("%d", &dan);
switch(dan) {
             case 1: printf("ponedeljak"); break;
             case 2: printf("utorak"); break;
             case 3: printf("sreda"); break;
             case 4: printf("cetvrtak"); break;
             case 5: printf("petak"); break;
             case 6: printf("subota"); break;
             case 7: printf("nedelja"); break;
             default: printf("uneli ste pogresan broj");
}
}
```

Program ce nam za uneti broj 3 ispisati 'sreda'
Evo jos jednog primera koriscenja switch operatora gde vise slucajeva (case-ova) ima isti rezultat. Program koji za uneti redni broj meseca ispisuje koliko mesec ima dana, sa podrskom za prestupnu godinu.(1 = januar ili sijecanj)

```
#include <stdio.h>
main() {
int mesec;
char ch;
printf("unesi redni broj meseca:\n");
```

```
scanf("%d", &mesec);
fflush(stdin);
switch(mesec) {
             case 1: case 3: case 5: case 7: case 8: case 10: case 12:
             printf("31 dan u mesecu"); break;
             case 4: case 6: case 9: case 11:
             printf("30 dana u mesecu"); break;
             case 2: printf("da li je godina prestupna d/n?\n");
                   scanf("%c", &ch);
                   if(ch == "D" || ch == "d") printf("29 dana u mesecu");
                   else printf("28 dana u mesecu"); break;
             default: printf("unet je pogresan broj");
}
}
```

Program ima 3 grananja, mesece koji imaju 31 dan, mecese koji imaju 30 dana i trece grananje mesec koji ima 28 ili 29 dana u zavisnosti od toga da li je godina presupna ili ne. Evo i izlaza programa:

```
unesi redni broj meseca:
5
31 dan u mesecu
```

Takodje, ako unesemo broj 2 odnosno drugi mesec, dolazimo do if grananja i to izgleda ovako:

```
unesi redni broj meseca:
2
da li je godina prestupna d/n?
d
28 dana u mesecu
```

Koristio sam jednu komandu, koju do sada nisam objasnjavao, a to je fflush(stdin); Ovo sluzi da se ocisti buffer kako ne bi doslo do mesanja inputa, i primeticete da sam stavio izmedju dve scanf() funkcije. Ovo treba koristiti kako ne bi imali problema mada se problemi javljaju samo kod char tipa podataka.

WHILE I DO WHILE CIKLUSI:

Ciklus je niz naredbi koji se izvrsava odredjeni broj ponavljanja. Svaki ciklus ima izlazni kriterijum, nakon koga prestaje ponavljanje naredbi. Imamo 2 vrste WHILE ciklusa, sa preduslovom i postuslovom. U principu imaju isti princip rada a razliku cemo primetiti u daljem tekstu.

- sa preduslovom: WHILE
- sa postuslovom: DO WHILE

Sintaksa za ciklus WHILE:

```
while(uslov) {
             naredba1;
             naredba2;
             naredba3;
```

```
}
```

Sintaksa za ciklus DO WHILE:

```
do {
              naredba1;
              naredba2;
              naredba3;
} while(uslov);
```

Kao sto vidimo razlika je sto kod WHILE ciklusa upisujemo prvo uslov pa zatim naredbe i zbog ovoga se zove ciklus sa preduslovom, dok se kod DO WHILE ciklusa prvo napise DO pa naredba ili niz naredbi, a zatim se ispisuje uslov ciklusa i zato se zove ciklus sa poduslovom. Programeri u vecini slucajeva koriste WHILE ciklus tako da cu u vecini slucajeva koristiti njega, ali princip rada je isti. Ciklus se ponavlja sve dok uslov ima vrednost tacno(eng. TRUE) odnosno, dok je razlicit od 0. Treba paziti da je uvek moguc izlaz iz ciklusa da on ne bi postao beskonacan. Evo primera WHILE ciklusa. Napisacemo program koji nam omogucava da unosimo brojeve sa tastature sve dok ne unesemo 0 ili negativan broj, odnosno, program koji nam omogucava unosenje samo pozitivnih brojeva (brojeva vecih od 0).

```
---------------------------------------------------------------------------
#include <stdio.h>
main() {
int broj=1;
while(broj>0) {
              printf("unesi neki broj ili 0 za izlaz: ");
              scanf("%d", &broj);
              printf("uneli ste %d\n", broj);
}
}
---------------------------------------------------------------------------
```

Kada pokrenemo program, ispisace nam se na ekranu poruka "unesi neki broj ili 0 za izlaz" nakon cega program ceka za unos sa tastature. Unesemo neki broj, npr. 5 i pritisnemo enter. Na ekranu nam je ispisana poruka "uneli ste 5" i ponovo poruka za unosenje broja. Posto smo videli da program radi, unesimo 0, program ispisuje poruku "uneli ste 0" i zavrsava se.
Na pocetku smo definisali da je varijabla broj=1 da ne bi doslo do zabune i uzimanja bilo koje vrednosti iz memorije.

Evo istog programa samo napisanog koriscenjem DO WHILE ciklusa:

```
---------------------------------------------------------------------------
#include <stdio.h>
main() {
int broj=1;
do {
              printf("unesi neki broj ili 0 za izlaz: ");
              scanf("%d", &broj);
              printf("uneli ste %d\n", broj);
} while(broj>0);
}
---------------------------------------------------------------------------
```

Kao sto vidimo, razlika je minimalna. Treba napomenuti da kod DO WHILE ciklusa nakon komande while(uslov) moramo staviti ; dok kod WHILE ciklusa ne stavljamo ;

Kako bi bolje shvatili WHILE ciklus, odradicemo jedan primer u kome cemo koristiti i brojac. Program koji ispisuje poruku na ekranu 10 puta.

```
#include <stdio.h>
main() {
int brojac=0;
while(brojac<10) {
             printf("ovo je neka poruka\n");
             brojac++;
}
}
```

Brojac krece od 0. Desavanja u ciklusu: ispise poruku, uveca brojac za 1, ispise poruku, uveca brojac za 1, i to sve dok je ispunjen uslov ciklusa a to je da je brojac<10, sto znaci da kada brojac bude bio 9, to ce biti zadnji put da se ciklus izvrsio (jer je krenuo od 0). Isti efekat bi dobili i da smo definisali pocetnu vrednost brojacu 1, i da smo kao uslov stavili brojac<=10 sto ce ukljuciti i slucaj kada je brojac=10.

Beskonacni ciklusi:

Za slucaj da zatrebaju ili da se susretnete sa njima, da znate o cemu se radi. Da bi ovo uspelo, potrebno je postaviti uslov koji ce uvek biti tacan. Evo nekoliko primera beskonacnih ciklusa (pisacu samo telo ciklusa, a ne ceo program jer je ostatak nebitan)

```
while(1) {
             printf("ovo ce se ispisivati sve dok nasilno ne ugasim program\n");
}
```

Kao sto sam napomenuo, potrebno je da je uslov uvek tacan, 1 je uvek TRUE, tako da je ciklus beskonacan. Evo jos jednog primera pogresno postavljenog uslova:

```
#include <stdio.h>
main() {
int i;
while(i>0) {
             printf("ovo je neki tekst\n");
             i++;
}
}
```

Petlja je takodje beskonacna i ispisivace 'ovo je neki tekst' sve dok nasilno ne ugasimo program. Toliko o beskonacnim petljama, evo jos koji primer kako bi lakse shvatili koriscenje ciklusa. Program koji ispisuje brojeve od 1 do N (broj N korisnih unosi sa tastature)

```
#include <stdio.h>
main() {
int n, broj=1;
```

```
printf("unesi broj n: ");
scanf("%d", &n);
while(broj<=n) {
             printf("%d\n", broj);
             broj++;
}
}
```

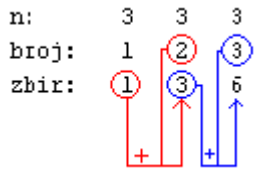
U kodu vidimo da ce se ciklus izvrsavati sve dok je varijabla broj manja ili jednaka od varijable n. U telu svakog ciklusa varijabla broj ispise svoju prethodnu vrednost i poveca se za 1.
Evo jos 1 program cisto za vezbu. Program koji sabira sve brojeve od 1 do N (broj N korisnik unosi preko tastature)

```
#include <stdio.h>
main() {
int n, broj=1, zbir=0;
printf("unesi broj n: ");
scanf("%d", &n);
while(broj<=n) {
             zbir=zbir+broj;
             broj++;
}
printf("zbir brojeva od 1 do %d je %d\n", n, zbir);
}
```

Program se ponasa isto kao i prethodni, s jednom razlikom, ne ispisuje brojeve na ekran, vec ih sabira sa prethodnom vrednosti varijable zbir. Znaci, na pocetku zbir=0, broj=1, i da kazemo da je korisnik uneo broj 3. Evo kako izgleda tablica brojeva:

```
n:                  3     3     3
broj:         1     2     3
zbir:         1     3     6
```

Komandom 'zbir=zbir+broj;' smestamo zbir stare vrednosti promenljive zbir i promenljive broj, u novu vrednost promenljive zbir. Posto zvuci dosta konfuzno napravio sam malu semu kako bi lakse shvatili:



OPERATOR CIKLUSA FOR:

Operator ciklusa FOR omogucuje bolji zapis ciklusa WHILE, ali nam i on, kao i WHILE sluzi za ponavljanje naredbi odredjeni broj puta.

Sintaksa:

```
for( inicijalizacija ; uslov ; korak ) {
            naredba1;
            naredba2;
            naredba3;
}
```

Kroz FOR ciklus ili petlju, menja se promenljiva koju definisemo u delu koji se naziva inicijalizacija, tu joj i ujedno damo i pocetnu vrednost. U drugom delu FOR petlje, definisemo uslov, koliko ce se puta izvrsiti naredbe unutar FOR petlje, i u trecem delu, korak, definisemo put stizanja od pocetne vrednosti do izvrsenja uslova. Da ne bih davio mnogo teorijom, evo kako sve to izgleda.

```
for( i=0 ; i<10 ; i++ )
```

Prvo smo definisali koju cemo promenljivu koristiti i dali joj pocetnu vrednost (i=0), u drugom delu smo definisali uslov koji znaci da ce se petlja pokretati iznova i iznova sve dok je i < 10 (i manje od 10) i u trecem delu, kako doci od prvog koraka do drugog, i++, nakon svakog prolazenja kroz FOR petlju, promenljiva i ce se uvecati za 1. Evo primene FOR petlje na konkretnom zadatku. Napisacemo program koji ispisuje neku poruku 10 puta.

```
-------------------------------------------------------------------------------
#include <stdio.h>
main() {
int i;
for(i=0;i<10;i++)
            printf("poruka\n");
}
-------------------------------------------------------------------------------
```

Mogao bih da dodam da u koliko imamo samo jednu naredbu koju stavljamo u FOR petlju nije potrebno da je stavljamo unutar {} zagrada, ali ako ih imamo vise, obavezno je staviti ih u zagrade.
Evo par primera kako bi lakse shvatili FOR petlje. Napisacemo program koji ce nam ispisati engleski alfabet (a,b,c,d...z).

```
-------------------------------------------------------------------------------
#include <stdio.h>
main() {
char ch;
for(ch='A';ch<='Z';ch++)
            printf("ASCII kod za %c je %d\n", ch, ch);
}
-------------------------------------------------------------------------------
```

Kada pokrenemo program, dobicemo sledeci ispis na ekranu:

```
-------------------------------------------------------------------------------
ASCII kod za A je 65
ASCII kod za B je 66
ASCII kod za C je 67
ASCII kod za D je 68
...
ASCII kod za Z je 90
-------------------------------------------------------------------------------
```

Kao sto vidimo, koristili smo promenljivu tipa char i kroz FOR petlju izredjali sve kombinacije pocevsi od A pa sve do Z tako sto smo promenljivoj ch dali pocetnu vrednost 'A', uslov je bio sve dok ch nije manje ili jednako sa 'Z' (ch<='Z') i nakon svakog prolaza kroz petlju, ch smo uvecavali za 1 (ch++). Ovaj isti program smo mogli da odradimo i na sledeci nacin:

```
---------------------------------------------------------------------------
#include <stdio.h>
main() {
int ch;
for(ch=65;ch<=90;ch++)
             printf("ASCII kod za %c je %d\n", ch, ch);
}
---------------------------------------------------------------------------
```

Sada cemo napisati program koji racuna zbir brojeva koje korisnik unosi:

```
---------------------------------------------------------------------------
#include <stdio.h>
main() {
int n, broj, i, suma=0;
printf("koliko brojeva zelis da saberes?\n");
scanf("%d",&n);
for(i=1;i<=n;i++) {
             printf("Unesi %d. broj\n", i);
             scanf("%d", &broj);
             suma=suma+broj;
}
printf("suma unetih brojeva je: %d", suma);
}
---------------------------------------------------------------------------
```

Kao sto vidimo iz koda, nakon pokretanja programa, korisnika prvo pitamo koliko brojeva zeli da sabere, taj broj koji unese je uslov, odnosno, do tog broja ce se izvrsavati FOR petlja. Nakon unesenog broja, program trazi korisniku da unese jedan po jedan broj i nakon unetog poslednjeg broja, izbaci poruku o konacnoj sumi.
sada cemo napisati rogram koji ispisuje sve brojeve od 0 do 99, ali koriscenjem 2 FOR petlje.

```
---------------------------------------------------------------------------
#include <stdio.h>
main() {
int jedinica, desetica;
for(desetica=0;desetica<10;desetica++)
             for(jedinica=0;jedinica<=9;jedinica++)
                   printf("%d\n",10*desetica+jedinica);
}
---------------------------------------------------------------------------
```

Zasto sam ovo uradio preko dve FOR petlje, a ne preko jedne, objasnicu kasnije, sada da objasnim princip rada ovog programa. Prvo da objasnim princip rada dve FOR petlje. Kao sto vidimo na primeru, desetica krece od 0 i ide do 9, jedinica isto, samo sam ih malo drugacije napisao cisto kako bih pokazao dva nacina zapisa istoga ('broj<10' je isto sto i 'broj<=9'). E sad, prva vrednost za deseticu je 0 i ona se nece promeniti sve dok se unutrasnja FOR petlja kompletno ne izvrsi, sto znaci, desetica=0, jedinica ce izredjati svoje vrednosti od 0-9, pa ce se tek onda desetica povecati na 1, a jedinica ce se opet izredjati od 0-9 i sve tako dok desetica ne bude 9, i jedinica 9.

```
--------------------------------------------------------------------------------
desetica=0 | jedinica=0 1 2 3 4 5 6 7 8 9
desetica=1 | jedinica=0 1 2 3 4 5 6 7 8 9
desetica=2 | jedinica=0 1 2 3 4 5 6 7 8 9
....
desetica=9 | jedinica=0 1 2 3 4 5 6 7 8 9
--------------------------------------------------------------------------------
```

Da objasnim i ispis(10*desetica+jedinica). Ovo je malo matematike. Izvrsava se ovako:

```
--------------------------------------------------------------------------------
10*0 + 0 = 0
10*0 + 1 = 1
10*0 + 2 = 2
.....
10*0 + 9 = 9
10*1 + 0 = 10
10*1 + 1 = 11
.....
10*9 + 7 = 97
10*9 + 8 = 98
10*9 + 9 = 99
--------------------------------------------------------------------------------
```

Iskoristio sam ovaj lagani primer kako bih objasnio kako se koristi petlja u petlji. Evo sada primera koji ce se malo teze odraditi preko jedne FOR petlje, pa koristimo tri. Napisacemo program koji ispisuje sve trocifrene brojeve kod kojih je cifra desetica manja od 5, a cifra jedinica neparna(1,3,5,7,9).

```
--------------------------------------------------------------------------------
#include <stdio.h>
main() {
int i,j,k;
for(i=0;i<10;i++)
              for(j=0;j<5;j++)
                    for(k=1;k<10;k+=2)
                          printf("%d\n",100*i+10*j+k);
}
--------------------------------------------------------------------------------
```

Radi na istom principu kao i prethodni primer. i - stotine, j - desetice, k - jedinice posto smo rekli samo trocifrene brojeve, i ima opseg od 1-9, posto smo rekli da cifra desetice mora da bude manja od 5, j ima opseg 0-4, i da cifra jedinice bude neparna k mogu da budu sledeci brojevi 1,3,5,7,9.

```
--------------------------------------------------------------------------------
i=1 ; j=0 ; k=1 | 100*1 + 10*0 + 1 = 101
i=1 ; j=0 ; k=3 | 100*1 + 10*0 + 3 = 103
i=1 ; j=0 ; k=5 | 100*1 + 10*0 + 5 = 105
.....
i=1 ; j=1 ; k=1 | 100*1 + 10*0 + 1 = 111
i=1 ; j=1 ; k=3 | 100*1 + 10*0 + 3 = 113
.....
i=2 ; j=0 ; k=1 | 100*2 + 10*0 + 1 = 201
i=2 ; j=0 ; k=3 | 100*2 + 10*0 + 1 = 203
.....
```

```
i=9 ; j=4 ; k=7 | 100*9 + 10*4 + 7 = 947
i=9 ; j=4 ; k=9 | 100*9 + 10*4 + 9 = 949
```

Takodje je moguce u jednoj FOR petlji staviti vise varijabli da se menjaju i prolaze cikluse. Evo primera kako bi lakse shvatili. Napisacemo program koji ispisuje dve kolone brojeva, jedna kolona ce biti od 0 do 30, a druga od 30 do 0.

```
#include <stdio.h>
main() {
int a,b;
for(a=0,b=30;a<=30,b>=0;a++,b--)
             printf("%d | %d\n", a, b);
}
```

Nakon sto pokrenemo program, dobicemo sledeci ispis na ekranu.

```
0 | 30
1 | 29
2 | 28
....
28 | 2
29 | 1
30 | 0
```

Kao sto vidimo, program radi bas ono sto smo i hteli, leva kolona se uvecava, desna se umanjuje. Ovo smo postigli tako sto smou istoj FOR petlji stavili dve varijable koje ce uporedo da menjaju vrednosti, odnosno, u jednom ciklusu petlje, obe promenljive ce promeniti vrednost. Promenljive smo samo odvajali zarezom ',' .

NAREDBA BREAK:

Naredba BREAK sluzi nam za nasilno izlazenje iz ciklusa, bilo WHILE, DO WHILE ili FOR. Evo primera kako bi lakse shvatili.

```
#include <stdio.h>
main() {
int i;
for(i=0;i<100;i++) {
             if(i==50) break;
             printf("%d\n", i);
}
}
```

Kada pokrenemo program, krenuce da ispisuje brojeve pocevsi od 0 pa na dalje... Prvobitni cilj mu je 99 (i<100), ali zbog sledece komande
if(i==50) break;
mi smo cilj promenili i kada i bude 50 (i=50) program ce iskociti iz petlje, i stati sa ispisom brojeva. Posto je if pre printf() funkcije, program nece ispisati broj 50, vec ce nakon ispisivanja broja 49 zavrsiti svoje izvrsavanje.
Takodje mozemo koristiti BREAK naredbu za izlazak iz beskonacne petlje. Evo primera.

```
--------------------------------------------------------------------------------
#include <stdio.h>
main() {
int i;
while(1) {
             if(i==10) break;
             printf("-------------------------\n");
             i++;
}
}
--------------------------------------------------------------------------------
```

Zapoceli smo beskonacnu petlju 'while(1)', ali smo u nju ubacili brojac ili korak (i++), sto ce nakon svakog ciklusa povecati promenljivu i za 1. U svakom ciklusu ispitujemo da li je i=10. U koliko nije, nastavlja sa izvrsavanjem i povecavanjem za 1, a ako jeste, dolazi do BREAK naredbe i tu izlazi iz petlje. Program ce u 10 redova ispisati ove crtice "------..."

FUNKCIJE:

Funkcije su deo koda koji cesto koristimo. Do sada smo se susretali sa nekim funkcijama main(), printf(), scanf(), fflush(). One nam omogucavaju podelu programa na podprograme. Takodje u koliko imamo neki proces koji cemo ponavljati vise puta u programu, korisno je da napisemo funkciju, i da to koristimo samo pozivanjem funkcije, sto nam znatno skracuje kod i vreme. Na pocetku teksta sam govorio o tipovima funkcija, tako da necu ponavljati, mada u koliko ste zaboravili, pozeljno je da se vratite na pocetak i procitate taj deo ponovo.

Definisanje funkcije:

```
--------------------------------------------------------------------------------
tip_rezultata ime_funkcije(argumenti) {
             telo_funkcije;
             }
--------------------------------------------------------------------------------
```

Poziv funkcije se vrsi navodjenjem njenog imena iza kojeg ide niz argumenata u zagradama.
Funkcije mogu biti pisane pre ili posle main() funkcije ali ako su iza main funkcije, moramo ih deklarisati pre, kucajuci tip funkcije, ime funkcije i tipove argumenata.

Primer pisanja funkcije pre main() funkcije:

```
--------------------------------------------------------------------------------
int funkcija(int x) {
             naredba1;
             naredba2;
             }
main() {
             naredba3;
             naredba4;
}
--------------------------------------------------------------------------------
```

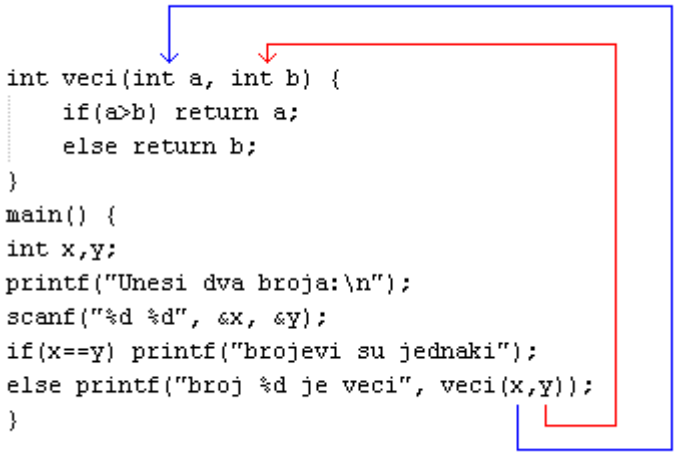
Primer definisanja funkcije pre main() funkcije, a pisanja posle.

```
-------------------------------------------------------------------------------
int funkcija(int);
main() {
naredba1;
naredba2;
}
int funkcija(int x) {
naredba3;
naredba4;
}
-------------------------------------------------------------------------------
```

Kao sto mozemo da primetimo, u drugom primeru, u deklarisanju funkcije, kao argument stoji samo tip argumenta, ali ne i njegovo ime. U koliko imamo vise argumenata, odvajamo ih zarezom ',' . Takodje da napomenem da argumente koje funkcija prima definisemo unutar zagrada, tako da nije potrebno da ih definisemo u telu funkcije. Da predjemo na primere kako bi lakse shvatili. Napisamo program koji racuna koji je od dva uneta broja veci, sa izdvojenom funkcijom za proveru koji je od dva broja veci.

```
-------------------------------------------------------------------------------
#include <stdio.h>
int veci(int a, int b) {
             if(a>b) return a;
             else return b;
}
main() {
int x,y;
printf("Unesi dva broja:\n");
scanf("%d %d", &x, &y);
if(x==y) printf("brojevi su jednaki");
else printf("broj %d je veci", veci(x,y));
}
-------------------------------------------------------------------------------
```

Ako malo bolje zagledate kod, primeticete da u argumentima funkcije veci() stoji 'int a, int b', a iz main() funkcije smo pozvali funkciju veci(x,y). Ovo je zato sto su varijable lokalne, odnosno, vaze samo unutar funkcije u kojoj su deklarisane. Pozivanjem funkcija, bitan redosled, ali ne i da se imena varijabli poklapaju. Pogledajte sledecu sliku:



```
int veci(int a, int b) {
    if(a>b) return a;
    else return b;
}
main() {
int x,y;
printf("Unesi dva broja:\n");
scanf("%d %d", &x, &y);
if(x==y) printf("brojevi su jednaki");
else printf("broj %d je veci", veci(x,y));
}
```

Napisacemo program koji sadrzi funkciju za ispis celog broja, i napisacu je sa funkcijom napisanom posle main() funkcije cisto kako bi videli kako to izgleda.

```
#include <stdio.h>
void ispisi(int);
main() {
int broj;
printf("Unesi broj\n");
scanf("%d",&broj);
ispisi(broj);
}
void ispisi(int br) {
             printf("%d", br);
}
```

Primer sam prikazao ga cisto zbog koriscenja drugog tipa funkcije i zapisa funkcije na drugom mestu.
Program koji sabira sve brojeve od 1 do N, korisreci funkciju suma()

```
#include <stdio.h>
int suma(int n) {
int suma=0, i;
if(n<0) return -1;
if(n==0) return 0;
for(i=1;i<n;i++)
             suma=suma+i;
return suma;
}
main() {
int sum, n;
printf("Unesi broj N:\n");
scanf("%d",&n);
sum=suma(n);
printf("Suma svih brojeva od 1 do %d je: %d", n, sum);
}
```

Prvo cu objasniti pozivanje funkcije, a zatim samu funkciju. Kao sto vidimo rezultat funkcije smestamo u promenljivu sum 'sum=suma(n);' a promenljiva n broj-granica do kog cemo vrsiti sumu brojeva. Funkcija suma() prima jedan broj kao argument, zatim ga ispituje, i ako je u odgovarajucem opsegu, a to je n>0 vrsi sabiranje, u suprotnom vraca vrednosti -1 i 0 u zavisnosti da li je n<0 ili n=0. Sa FOR petljom uzimamo sve brojeve od 1 do n, sabiramo ih i smestamo u promenljivu suma, nakon cega vracamo vrednost promenljive suma kako bi mogla da se prikaze u glavnoj funkciji.
Jos jedan primer za kraj kako bi shvatili kontrolisanje povrathinh informacija. Napisacemo program sa odvojenom funkcijom koja racuna da li je broj prost ili ne. Kako bi ovo uradili prvo moramo da znamo kada je broj prost. Broj je prost kada je deljiv samo samim sobom i brojem 1. Ako je broj deljiv samim sobom, 1, i makar jos jednim brojem, onda taj broj nije prost. Za resavanje ovog problema trebace nam malo znanja matematike, ali sve cu objasniti. Evo programa.

```
#include <stdio.h>
int prost (int n) {
             int i,j;
             for (i=2; i<n/2; i++)
                   if ((n%i)==0) return 0;
             return 1;
}
main() {
int br;
scanf("%d", &br);
if(prost(br)==0) printf("broj nije prost");
else printf("broj je prost");
}
```

Da se bacimo na objasnjavanje. Funkcija prost() nam sluzi za testiranje jednog celog broja. Imamo FOR petlju koja ima opseg od 2 do n/2. Pocinjemo od broja 2 jer je svaki broj deljiv sa 1, pa nam to ne treba, a zavrsavamo se sa n/2, jer ako delimo n sa bilo kojim brojem vecim od n/2, rezultat ce biti manji od 2, odnosno, da kazemo to obicnim recima, u koliko ne otkrijemo da li je broj prost deljenjem od 2 do polovine tog broja, dalje nema svrhe traziti. Funkcija prost() proverava broj i u koliko otkrije da je pri deljenju sa nekim brojem, ostatak ostao 0, odnosno da je broj deljiv tim brojem, vraca rezultat 0, u suprotnom vraca 1. Toliko o prost() funkciji. Iz main() funkcije ispitujemo koji nam rezultat vraca funkcija prost(). U koliko nam vrati broj 0, znamo da broj nije prost, jer je funkija prost() nasla neki broj s kojim je nas broj deljiv bez ostatka, a u suprotnom, odnosno ako nam vrati 1, znamo da je broj prost, i te poruke ispisujemo na ekranu.

OUTRO:

Poceo sam da pisem tekst kao neki osnovni o programiranju u C-u, ali se ispostavilo da kako sam napisao jednu stvar, jos dve su mi pale na pamet, nakon par dana pisanja zavrsilo se sa ovim, malo povecim tekstom. Nadam se da sam vam bar malo objasnio osnove programiranja u C-u. U koliko vam posle citanja ovog teksta neke stvari i dalje nisu jasne, pitajte nekoga, procitajte jos tekstova, ali NE odustajte. Programiranje je lepa i korisna stvar. Ne morate koristiti C, koristite neki drugi programski jezik blizi vasim potrebama, ali kad se nauce osnove, onda je svaki jezik lagan. Korisno je izabrati C za ucenje programirajna, kao sto sam spomenuo na pocetku zbog jednostavne sintakse. Sta vam sada preostaje, nista drugo nego da

citate, ucite, vezbate. Sve je u vezbanju. Srecno sa programiranjem. Pozdrav.

##EOF##